Device = UDISK.SYS 180 128 64

For Commodore 64™

IEA

Instant
Editor
Assembler

Personal Comments by Jim Gracely, Technical Editor, Commodore Magazine

The IEA Instant Editor Assembler package is an invaluable asset to the beginning machine language programmer. The package contains an excellent editor/assembler system, a full featured monitor and a walk program with variable step speed and the ability to add break points. All three of these programs may be resident in memory at the same time. The individual programs are easy to understand, easy to use and work quickly and efficiently.

The package as a whole is quite professionally written. The Disk which contains the main programs also contains six programs assisting in the use of these programs and more than a dozen additional utility and example programs. One additional feature of the manual is a technical assistance number to aid you with any problems or questions you might have.

"I would have to recommend the IEA Instant Editor Assembler, the BEST VALUE-FOR-PRICE development package on the market!" **Jim Gracely**

"IEA is a nice simple assembler, good for beginners at an UNBELIEVABLE PRICE!"
Jim Butterfield, author of "Machine Language" Programming for the Commodore 64 and Other Computers."

Robin's Software

Box 20220 • Bloomington, MN 55420
Phone (612) 944-8654

IEA©

Instant Editor Assembler

For Commodore 64 T.M.

IEA Instant Editor Assembler

for use with the COMMODORE 64™
with disk drive

Robin's Software
Box 20220 Bloomington, MN 55420
Phone (612) 944-8654

Programmed by Mark Robin

SECOND EDITION

SECOND PRINTING-1984

CONTENTS

INTRODUCTION

Congratulations, you have just purchased the fastest, most versatile EDITOR ASSEMBLER on the market. IEA is capable of assembling up to 17k source files in less than 4 seconds. Fourteen commands are added to the BASIC editor to create source files that can be assembled by IEA. Useful commands include FIND, DELETE, RENUMBER, AUTO, UNNEW, APPEND, BLOCK SAVE, HEX TO DECIMAL, and DECIMAL TO HEX to name a few.

IEA includes MICROMON one line assembler disassembler for disassembling and editing binary files. Also included is a special WALK DEBUGGING PACKAGE. This program can be used to walk through GAME programs because it uses only the top two lines of the screen to display the registers, flags, program counter, op code, and stack pointer. Breakponts and variable speeds for slow motion execution can be set.

The three main programs (IEA/SYS, MONITOR, WALK) can all reside in memory at the same time. That means mistakes can be corrected immediatly, reassembled and tested without disk access.

Sample programs are included with all the subroutines to assist the programmer in understanding the fundemental features of 6510 assembly language.

An assembly language hotline has been established with the advent of IEA to assist programmers (beginners to professionals) with information specific for the Commodore 64 microcomputer. To contact this hotline, phone 612-944-8654 during the hours 6:00 p.m. to 10:00 p.m. 7 days a week.

SOFTWARE CONTAINED ON DISKETTE

IEA/SYS Instant Editor Assembler is a 9k machine language program located in memory at $4A00-$6FFF HEX.

MONITOR Monitor is a machine language program located at $9000 or $C000 HEX.

SMP1/IEA SAMP1/IEA is a SOURCE text file illustrating the use of all Fourteen EDITOR Commands. Use this file with "Getting Started with IEA".

SMP2/IEA SAMP2/IEA is a SOURCE text file illustrating the use of all eight PSEUDO OP CODES.

WALK WALK is a machine language program to allow single stepping through a machine language program. This WALK uses only the top 2 lines of the monitor to display the PROGRAM COUNTER, REGISTERS, and FLAGS. This feature allows the programmer to walk through GAME programs without disturbing the screen. WALK is located at $4000 or $C000 HEX.

BAS/IEA BAS/IEA is a SOURCE text file to show how to pass information between BASIC and MACHINE LANGUAGE programs, ZERO PAGE MEMORY, and ROM Routines.

IEANOTES IEANOTES/TXT. Load this file and LIST to see Editor commands, Pseudo Op Codes and also any notes added after printing of this manual.

SPRITE SPRITE.SUBS/IEA is a source file with fourteen SPRITE subroutines that make controlling sprites easier than using BASIC.

ANDLOGIC AND.LOGIC/IEA describes the use of mnemonic AND.

EORLOGIC EOR.LOGIC/IEA describes the use of mnemonic EOR.

ORALOGIC ORA.LOGIC/IEA describes the use of mnemonic ORA.

INPUT INPUT.PRINT/IEA are two subroutines that can be used to INPUT Variables and PRINT them.

SOFTWARE CONTAINED ON DISKETTE (CONT)

STRING STRING.CMP/IEA is a subroutine that will compare two strings to see if they are equal.

PONG PONG/IEA is a sample GAME program using SPRITES.

HOW TO HOW TO WALK/TXT is a description of using the WALK DEBUGGING PROGRAM.

ASC-BIN ASC-BIN.ROM/IEA is a subroutine describing the use of the ASCII to BINARY ROM routine.

BIN-ASC BIN-ASC.ROM/IEA is a subroutine describing the use of the BINARY to ASCII ROM routine.

SAVE/IEA SAVE/IEA is a subroutine describing the KERNAL routine SAVE.

LOAD/IEA LOAD/IEA is a subroutine describing the KERNAL routine LOAD.

CURSOR CURSORFLASH/IEA is a subroutine to flash the cursor and wait for keyboard input. It returns with an ascii character in the Accumulator.

INPUT INPUT.FLASH/IEA is an input subroutine that can be used to input strings of data from the keyboard. It handles BACKSPACE,CURSOR right and left, and can receive character strings up to 255 bytes.

INSTR INSTR is a subroutine that can be used from a BASIC program to find the first occurance of a substring within a string.

SORT Shell SORT is a subroutine to sort a BASIC string array into alphabetical order.

TEXT EDITOR COMMAND FEATURES

.A	Assemble Source text file.
.Dn,n	Delete a range of lines.
.F/text/	Find and list all lines containing text.
.In	Set increment for AUTO and RENUMBER.
.I/	Turn AUTO OFF.
.K	Kill EDITOR wedge.
.Ln	List SOURCE text.(n=start line)
.R	Renumber text file.(increment set with .In)
.S	Save a block of memory. Syntax-[.S "FILENAME" C000 CFFF]
.Tn	Set Format Tab.
.X	List Hex address and Labels.
.U	UNNEW restores a lost text file.(BASIC File)
.M	Sets start of text pointers to end of text to allow a program to be loaded on top of an existing file.
$	Convert Hex to Decimal.
#	Convert Decimal to Hex.

The SOURCE text files are in EXACTLY the same format as BASIC SOURCE files. All BASIC commands will work normally with the EDITOR wedge in place. BASIC and ASSEMBLY programs can be EDITED at the same time. Use LOAD and SAVE commands to LOAD or SAVE SOURCE text files. Locate ASSEMBLY text on lower line numbers then BASIC programs and use the .EN PSEUDO OP to tell the assembler the end of text to assemble.

To Disable the EDITOR wedge, type .K
To Reenable the EDITOR wedge, type SYS 6*4096

IEA is compatible with HESmon cartridge.

IEA is compatible with the DOS wedge program supplied with commodore disk drives.

ERROR MESSAGES

SYNTAX 00000 — A line of text can not be assembled. Assembly stops at that line.

RANGE 00000 — A branch is out of range.

NO MATCH 00000 — :LABEL not defined.

OVERFLOW 00000 — Label Table overflow has occured. Maximum Labels - 400. Maximum Labels Referenced - 332.

DUPLICATE 00000 — Two :LABELS have the same name. This error is generated by the .X command only and not during assembly.

PSEUDO OP CODES

.DB — Define Byte. Stores Hex values in memory.
Syntax-[.DB $44 $55 $xx etc....]
or [.DB $4455xx etc....]

.DS — Define Space. Reserves an area of memory. Syntax-[.DS $00FF] allocates 255 bytes.

.DW — Define Word. Stores Ascii Values of text. Syntax-[.DW "THIS IS A STRING"]

.EN — End of program to assemble.

ORG — Sets the location counter. Syntax-[ORG $C000] Start assembling at $C000.

.SI — Set internal address of a label low byte first. Syntax-[.SI :LABEL] If LABEL = $C000 then the low byte $00,high byte $C0 would be stored in memory.

.XY — Set X= low byte, Y= high byte. Syntax-[.XY :LABEL] would generate the code LDX #$low LDY #$high, of the labels address.

***** — If an asterisk is the first character on a line it will be stored as a 00 byte in successive memory locations. Syntax-[00010 ***] stores 3 zero bytes.

.EQ — Assign a hexidecimal address to the label. Syntax-[00010 :LABEL .EQ $C000]

EXAMPLE OF .XY, .DW, .EQ, ORG, PSEUDO-OP CODES

```
00010             ORG $C000         ;START ASSEMBLY
00015 :CHROUT     .EQ $FFD2         ;EQUATE LABEL
00020 :MESSAGE                      ;LABEL
00030             .DW "HELLO"       ;DEFINE WORD
00035             *                 ;STORE 00 BYTE
00036 :ENTRY                        ;START OF PROG
00040             .XY :MESSAGE      ;GET LOW,HIGH
                                    ;ADDRESS IN THE
                                    ;X AND Y REGISTER
00050             STX $FB           ;STORE LOW BYTE
00060             STY $FC           ;STORE HIGH BYTE
00070             LDY #$00          ;INITIALIZE COUNTER
00080 :LOOP                         ;LABEL
00090             LDA ($FB),Y       ;GET LETTER IN A
00095             BEQ :RETURN       ;CHECK FOR 00
00100             JSR :CHROUT       ;OUTPUT CHAR
00110             INY               ;INCREMENT POINTER
00120             JMP :LOOP         ;DO NEXT LETTER
00121 :RETURN                       ;LABEL
00125             RTS               ;RETURN TO BASIC
00126 :END                          ;LABEL
00130             .EN               ;END OF ASSEMBLY
```

The program above will print the word 'HELLO' to the screen.
Type .A [press return] to assemble.
Type .X [press return] to see label table. It should look like this:

```
$FFD2 :CHROUT               $C000 :MESSAGE
$C006 :ENTRY                $C010 :LOOP
$C01B :RETURN               $C01C :END
```

The program starts at the label :ENTRY. The HEX address is to the left of the label table listing. To convert $C006 to a decimal number you can use the hex to decimal function like this:

Type: $C006 [press return]
IEA will respond with the decimal value
Type SYS 49158 [press return]

To make life easier add these lines to the program. Now to run, simply type 'RUN'.

```
00000 :SYS 49152:STOP
00011             JMP :ENTRY
```

The colon before the SYS command must be used, or IEA will give a SYNTAX ERROR. By using a JMP as the first command after the ORG, the SYS address will always be correct.

GETTING STARTED WITH IEA

IEA is supplied on a 5 1/4 Floppy disk for the Commodore 64 computer. The program can not be copied or backed up. If the IEA Master disk is destroyed, another Master copy can be purchased for a nominal fee of $5.00. Follow the instructions on the WARRANTY CARD.

LOADING IEA: Insert the disk in the disk drive.
TYPE: LOAD "IEA/SYS",8,1

The copyright notice should be displayed.
TYPE: NEW

Text EDITOR commands will only work if there is text in memory. If there is no text or if a .K(kill wedge) has been done, a syntax error will be displayed.

Load the sample program.
TYPE: LOAD "SMP1/IEA",8

To list the program.
TYPE: .L [PRESS ENTER]
The first full screen of text will be displayed on the screen. PRESS THE SPACE BAR to continue listing or press the BREAK key to return to command mode.

The command .T will set the amount of space between the line number and the op codes. The TAB should be set for the maximum length of the labels to be used.

To reset the TAB
TYPE: .T number (0-10) [PRESS ENTER]

To list to the printer, use OPEN4,4:CMD4 or the command you usually use to list a BASIC program.

Finding a string of text
TYPE: .F/MESSAGE/ [PRESS ENTER]
You should see lines 20 and 40 listed on the screen.

Assembling the source file
TYPE: .A [PRESS ENTER]
You should see IEA PASS 1 and IEA PASS 2
The label table is now in memory and you can use the .X command.

Label table listing
TYPE: .X
You will see the hex value and the labels name for all labels. Labels that use the + or - sign are listed at the end.

If you see the message DUPLICATE LABEL line, this is a warning that the file contains duplicate labels. You should change the label to another name.

To find all references to a duplicate label, use the .F command like this ".F/ :labelsname/". Notice the space before the colon. Only lines that reference labels will be listed.

To turn AUTO line numbering ON
TYPE: .I number (1-1000)
The .I command also sets the increment for the .R command.

To turn AUTO OFF
TYPE: .I/
The increment is still in memory for the .R command.

To renumber program
TYPE: .R
Make sure that line numbers do not exceed 64000. If they do, reset the increment to a lower number and use .R to renumber again.

To Delete lines of text
TYPE: .Dstart,end
Start and End should be line numbers in the text file. If Start or End are not found in the file, a Syntax error will be displayed.

To save an assembled machine language program.
TYPE: .S "program name" start end
Start should be the ORG address.
End should be the last address assembled.
Look at the label table listing for the label :END.
This address is the LAST address of the machine language program.

You should always end your assembly programs like this if you want to know the last address used:
line number :END
line number .EN

LABELS

Labels may be any length and must always start with a colon ":". A Label after an op code may be an expression using the "+" or "-" sign. Maximum of 255 bytes. $FF Hex.

Some examples of valid label expressions
```
LDA :DATA+$01,Y
LDA :DATA-$01
LDA :DATA-$FF,X
```

MICROMON MONITOR

COMMAND	DESCRIPTION	SYNTAX
A	Simple Assembler	.A C000 LDA #$00
C	Compare Memory	.C C000 CFFF 1000
D	Disassembler	.D C000 .D C000 C010
E	Exit Monitor	.E
F	Fill Memory	.F C000 CFFF 00
G	Go, Run	.G C000
H	Hunt Memory	.H C000 CFFF 00 0A 01 .H C000 CFFF 'TEXT
K	Kill Monitor	.K
L	Load	.L "PROGRAM",08
M	Memory Display	.M C000 .M C000 C020
N	New Locator	.N C000 CFFF 1000 3000 4000
O	Branch Offset	.O C000 C005 03
P	Toggle Output	
R	Register Display	.R
S	Save	.S "PROGRAM",C000,CFFF,08
T	Transfer Memory	.T C000 CFFF 3000
X	Exit to Basic	.X
Z	Change Character Sets	.Z
$	Hex to Decimal	$C000
-	Hex Subtraction	-C000 C001
&	Checksum	&C000 C010

SIMPLE ASSEMBLER
.A C000 LDA #$01
Assembles when you press the return key.
.A C000 A9 01 LDA #$01

COMPARE MEMORY
.C 1000 2000 3000
Compares memory from HEX 1000 to HEX 2000 to memory beginning at HEX 3000. Locations with unequal bytes will be printed on the screen.

DISASSEMBLER
.D C000 C020
Disassembles from address C000 to C020. The two or three bytes following the address can be modified by placing the cursor over the character to be changed. If the cursor is on the last line of the top of the screen, the disassembly can be scrolled by pressing the cursor key up or down.

FILL MEMORY
.F C000 CFFF 00
This command will fill memory from C000 to CFFF with 00.

GO COMMAND
.G C000
Starts execution at C000.
.G
Starts execution at the location contained in the PC register.

HUNT MEMORY
.H C000 CFFF 'TEXT
Hunt thru memory from C000 to CFFF for the ascii string TEXT .H C000 C500 0A 01 05
Hunt thru memory for a sequence of bytes 0A 01 05 and print the address where they occurred.

NEW LOCATOR
The NEW Command is used to relocate binary code.
If you assemble a program at C000 and you want to relocate it to 5000, first use the Transfer Memory command to transfer the code from C000-xxxx to 5000-xxxx. Then type .N 5000 xxxx 9000 C000 xxxx where xxxx is the end of the program.

You should only NEW LOCATE actual machine code and not DATA contained within the code. All DATA tables, strings etc... should be at the start or end of assembly code, that makes relocating a snap.

EXAMPLE USING THE .N COMMAND

```
   C000 LDA #$01
   C002 JMP $C000
.T C000 C004 5000
```

Transferred code at 5000

```
   5000 LDA #$01
   5002 JMP $C000
.N 5000 5004 9000 C000 C004
```

The 9000 is added to any three byte op code that addresses memory in the range C000-C004.

New code at 5000 ;C000 + 9000 = 5000 in location 5002.

```
   5000 LDA #$01
   5002 JMP $5000
```

REGISTER DISPLAY

```
.R
   PC   IRQ  SR AC XR YR SP
.; 0000 0401 02 02 00 FE FF
```

The contents of the register may be changed using the cursor and typing a new value.

SAVE

.S "PROGRAM",C000,CFFF,08
Save memory from C000 to CFFF file name PROGRAM.

TRANSFER MEMORY

.T C000 CFFF 3000
Transfer 4000 bytes of memory from C000 to CFFF to 3000.

There are two monitors on the master diskette. MONITOR$9000 and MONITOR$C000. The REGISTER Display will have the name MICR9000 or MICRC000 so you know which monitor you are using.

Loading the Monitor.

TYPE LOAD"MONITOR$9000",8,1
TYPE SYS9*4096
or
LOAD "MONITOR$C000",8,1
SYS 49152

Press the RESTORE key from BASIC to get to the monitor. If a BRK instrucion is executed, the monitor will take control.

See **WALK/SYS** for an explanation of single stepping or slow motion execution of Machine Language programs.

USING WALK

WALK is a program designed for debugging and checking program flow. By using only the top two lines on the video display, a programmer can walk through a machine language program that does graphic displays. There are four function keys.

F1 key Hold down to increase speed.
F2 key Hold down to decrease speed.
Space bar Hold down for slow motion. Press for single step.
B Press to set a breakpoint.
F7 Continue after break.
Pressing the BREAK key will exit the program.

Use WALK$4000 to walk through programs located in high RAM or ROM. Entry is SYS4*4096.

Use WALK$C000 to walk through programs located in low RAM or ROM. Entry is SYS 49152.

To run WALK$4000, type LOAD "WALK$4000",8,1 [PRESS ENTER]
TYPE SYS 4*4096

The program will ask for a start address in Hex. After typing the address, the program will display:

PC	SP	NVBDIZC	A	X	Y	OP
XXXX	XXXX	XXXXXXX	XX	XX	XX	XX

PC Address of the next instruction to be executed.
SP Stack Pointer.

PROCESSOR STATUS REGISTER "P"

N	Negative Flag.	1=Negative
V	Overflow Flag.	1=True
B	BRK Command Flag	1=BRK
D	Decimal Mode Flag.	1=True
I	IRQ Disable Flag.	1=Disable
Z	Zero Flag.	1=Result Zero
C	Carry Flag.	1=True

A REGISTER
X REGISTER
Y REGISTER

To set a breakpoint, press "B" and then answer the prompt: ENTER ADDRESS TO SET BREAK.......

SPRITE MOVING SUBROUTINES

On the next two pages are fourteen subroutines to make writing game programs that use SPRITES easy.

```
   '   SPRITE MOVING SUBROUTINES
-----------------------------------
00010          ORG $C000
00020 :A
00030          #
00040 :X
00050          #
00060 :Y
00070          #
00080 :MASK1
00090          .DB $0102040810204080
00100 :MASK2
00110          .DB $FEFDFBF7EFDFBF7F
00120 :PUTREG
00130          STA :A
00140          STX :X
00150          STY :Y
00160          RTS
00170 :PULLREG
00180          LDA :A
00190          LDX :X
00200          LDY :Y
00210          RTS
00220
   '
-----------------------------------
00230 :SMVL    ;SPRITE MOVE LEFT
00240          JSR :PUTREG
00250          TAX
00260          TAY
00270          LDA $D010
00280          AND :MASK1,X
00290          BNE :SET
00300          TXA
00310          ASL A
00320          TAX
00330          DEC $D000,X
00340          JSR :PULLREG
00350          RTS
00360 :SET
00370          TXA
00380          ASL A
00390          TAX
00400          DEC $D000,X
00410          LDA $D000,X
00420          CMP #$FF
00430          BNE :RETURN
00440          LDA $D010
00450          AND :MASK2,Y
00460          STA $D010
00470 :RETURN
00480          JSR :PULLREG
00490          RTS
00500
   '
-----------------------------------
00510 :SMVR    ;SPRITE MOVE RIGHT
00520          JSR :PUTREG
00530          TAX
00540          TAY
00550          LDA $D010
00560          AND :MASK1,X
00570          BEQ :NOTSET
00580          TXA
00590          ASL A
00600          TAX
00610          INC $D000,X
00620          JSR :PULLREG
00630          RTS
00640 :NOTSET
00650          TXA
00660          ASL A
00670          TAX
00680          INC $D000,X
00690          BNE :RETURN
00700          LDA $D010
00710          ORA :MASK1,Y
00720          STA $D010
00730          JSR :PULLREG
00740          RTS
00750
   ' TURN A SPRITE ON A=0-7 OR $FF
-----------------------------------
00760 :SPON
00770          JSR :PUTREG
00780          CMP #$FF
00790          BEQ :ALLON
00800          TAX
00810          LDA :MASK1,X
00820 :ALLON
00830          ORA $D015
00840          STA $D015
00850          JSR :PULLREG
00860          RTS
```


SPRITE.SUBS/IEA

```
00870
   ' TURN A SPRITE OFF A=0-7, OR $FF
-----------------------------------
00880 :SPOFF
00890          JSR :PUTREG
00900          CMP #$FF
00910          BEQ :ALLOFF
00920          TAX
00930          LDA :MASK2,X
00940          AND $D015
00950          STA $D015
00960          JSR :PULLREG
00970          RTS
00980 :ALLOFF
00990          LDA #$00
01000          STA $D015
01010          JSR :PULLREG
01020          RTS
01030
   '     SET SPRITE POINTERS
-----------------------------------
01040 :SPOINT ;A=SPRITE, X=BLOCK
01050          JSR :PUTREG
01060          AND #$07
01070          TAY
01080          TXA
01090          STA $07F8,Y
01100          JSR :PULLREG
01110          RTS
01120
   '      SPRITE COLOR
-----------------------------------
01130 :SPCOL   ;A=0-7, X=0-15
01140          JSR :PUTREG
01150          AND #$07
01160          TAY
01170          TXA
01180          STA $D027,Y
01190          JSR :PULLREG
01200          RTS
01210
   '       SPRITE UP
-----------------------------------
01220 :SPUP    ;A=0-7
01230          JSR :PUTREG
01240          AND #$07
01250          SEC
01260          ROL A
01270          TAX
01280          DEC $D000,X
01290          JSR :PULLREG
01300          RTS
01310
   '      SPRITE DOWN
-----------------------------------
01320 :SPDOWN
01330          JSR :PUTREG
01340          AND #$07
01350          SEC
01360          ROL A
01370          TAX
01380          INC $D000,X
01390          JSR :PULLREG
01400          RTS
01410
   ' EXPAND HORIZONTALLY
-----------------------------------
01420 :SPEXH
01430          JSR :PUTREG
01440          TAX
01450          LDA :MASK1,X
01460          ORA $D01D
01470          STA $D01D
01480          JSR :PULLREG
01490          RTS
01500
   '  EXPAND VERTICALLY
-----------------------------------
01510 :SPEXV
01520          JSR :PUTREG
01530          TAX
01540          LDA :MASK1,X
01550          ORA $D017
01560          STA $D017
01570          JSR :PULLREG
01580          RTS
01590
   '  UNEXPAND HORIZ.
-----------------------------------
01600 :SPUNH
01610          JSR :PUTREG
01620          TAX
01630          LDA :MASK2,X
01640          AND $D01D
01650          STA $D01D
01660          JSR :PULLREG
01670          RTS
```

```
01680
      '   UNEXPAND VERT.
-------------------------------------
01690 :SPUNV
01700          JSR :PUTREG
01710          TAX
01720          LDA :MASK2,X
01730          AND $D017
01740          STA $D017
01750          JSR :PULLREG
01760          RTS
01770
      '   MULTI-COLOR MODE SET
-------------------------------------
01780 :SPMCS
01790          JSR :PUTREG
01800          TAX
01810          LDA :MASK1,X
01820          ORA $D01C
01830          STA $D01C
01840          JSR :PULLREG
01850          RTS
01860
      '    MULTI-COLOR CLEAR
-------------------------------------
01870 :SPMCC
01880          JSR :PUTREG
01890          TAX
01900          LDA :MASK2,X
01910          AND $D01C
01920          STA $D01C
01930          JSR :PULLREG
01940          RTS


01970
      '     EXAMPLE SPRITE PROGRAM
--------------------------------------
01980 :LOC    .EQ $0380 ;CASSETTE BUFF
01990 :HIBIT  .EQ $D010
02000 :SOCOL  .EQ $D000 ;SPRITE#0 COL
02010 :SOROW  .EQ $D001 ;SPRITE#0 ROW
02020 :SWITCH
02030         .DB $01
02040 :ROW              ;START ROW
02050         .DB $60
02060 :COL              ;START COL
02070         .DB $18
02071 ;
02080 :EXAMPLE
02090          LDY #$3B  ;64 BYTES
02100          LDA #$FF  ;FILL ALL BITS
02110 :LP                  ;FILL MEM
02120          STA :LOC,Y
02130          DEY
02140          BPL :LP
02150          LDA #$00  ;SPRITE NUMBER
02160          JSR :SPON ;TURN ON
02170          LDX #$0E  ;LOC POINTER
02180          JSR :SPOINT ;SET POINTER
02190          LDA :COL   ;SET UP
02200          STA :SOCOL ;COL SPRITE#0
02210          LDA :ROW   ;SET UP
02220          STA $D001 ;ROW SPRITE #0
02230 :LP1
02240          LDA :SWITCH
02250          BMI :LEFT    ;IS IT MINUS
02260          LDA #$00
02270          JSR :SMVR ;MOVE RIGHT
02280          JMP :DELAY ;JUMP TO DELAY
02290 :LEFT              ;MOVE LEFT
02300          LDA #$00
02310          JSR :SMVL
02320 :DELAY
02330          LDA $D012  ;RASTER SCAN
02340          BNE :DELAY ;WAIT IF NOT 0
02350 :TESTO TEST IF ON RIGHT OF SCREEN
02360          LDA :HIBIT
02370          AND #$01  ;IS IT SET
02380          BEQ :TEST1 ; NO
02390          LDA :SOCOL ;COLUMN
02400          CMP #$41   ;END OF SCREEN
02410          BNE :LP1   ;NO
02420 :FLOP
02430          LDA :SWITCH
02440          EOR #$FF   ;FLOP BYTE
02450          STA :SWITCH
02460          BNE :LP1   ;NEVER 0
02470 ;         --------
02480 :TEST1 TEST IF ON LEFT OF SCREEN
02490          LDA :SOCOL
02500          CMP :COL   ;IS IT
02510          BNE :LP1   ;NO
02520          BEQ :FLOP  ;YES
02530 ;         --------
02540 ;THE ROUTINE ABOVE WILL BOUNCE
02550 ;A SPRITE ACROSS THE SCREEN.
```


IMMEDIATE ADDRESSING
Operand Format: #$hh

There are eleven commands that use the immediate addressing mode. They are:

ADC ADd with carry flag an 8 bit value.

AND AND the value in the Accumulator with an immediate 8 bit value.

CMP CoMPare the value in the accumulator with an immediate 8 bit value.

EOR Exclusive OR the accumulator register with an 8 bit value.

LDA LoaD the Accumulator with an 8 bit value.

ORA OR the Accumulator with an 8 bit value.

SBC SuBtract with Carry flag an 8 bit value from the accumulator.

CPX ComPare the X register with an 8 bit value.

CPY ComPare the Y register with an 8 bit value.

LDX LoaD the X register with an 8 bit value.

LDY LoaD the Y register with an 8 bit value.

Examples of ASSEMBLY and BASIC commands.

(Clear the video screen.)

```
PRINT CHR$(147)

LDA #$93  ;CLEAR HOME HEX ASCII CODE
JSR $FFD2 ;DO PRINT SUBROUTINE
```

(For next loop)

```
FOR X=1 TO 5 : NEXT

    LDX #$01  ;LOAD X WITH VALUE OF 1
:LP           ;LABEL START OF LOOP
    INX       ;INCREMENT THE X REGISTER
    CPX #$05  ;COMPARE X WITH VALUE OF 5
    BNE :LP   ;BRANCH IF NOT EQUAL TO :LP
```

ABSOLUTE ADDRESSING
Operand Format: $hhhh
Operand Format: :LABEL

There are twenty three commands that use the absolute addressing mode. Any absolute addressing mode may use either a hexidecimal or a label in its operand field. All absolute operands refer to an address in memory similar to a POKE address or PEEK address in BASIC.

The eleven commands listed under IMMEDIATE ADDRESSING may also be used in ABSOLUTE ADDRESSING mode. Here are the other twelve commands.

ASL Accumulator Shift Left is the english translation for this code. Although, in the absolute mode it has nothing to do with the accumulator. All 8 bits in the memory location pointed to by the absolute operand are shifted to the left one bit position. The high bit (bit 7) will be shifted to the carry flag. Also the low bit (bit 0) will be set to zero (0). This command has the effect of multyplying an 8 bit value by 2.

LSR Local Shift Right. The opposite of ASL. The high bit of the 8 bit value pointed to by the operand is set to 0, and the low bit is moved to the carry flag. This command has the effect of dividing an 8 bit value by 2.

ROL ROtate Left. This command works like the shift commands except that the carry flag is rotated into the low bit pointed to by the operand and then the high bit is moved to the carry bit.

ROR ROtate Right. The opposite of ROL. The carry flag is moved to the high bit position pointed to by the operand and the low bit is moved to the carry flag.

BIT test BITs in memory with a value in the accumulator. The accumulator is not affected by this command. This command performs two operations at the same time. First bit 7 is moved to the negative flag and bit 6 is moved to the overflow flag. Second, a simulated AND operation is performed and the result if equal, to zero sets the Z flag to logic 1. If the result of the simulated AND operation is not equal the Z flag is cleared to logic 0.

STA STore the Accumulator in a memory location.

STX STore the X register in a memory location.

STY STore the Y register in a memory location.

JMP JuMP to an absolute memory address.

JSR Jump to a SubRoutine at the absolute memory address.

Examples of ASSEMBLY and BASIC commands.

(8 bit multiply by 2)

```
POKE 49152,PEEK(49152)*2

ASL $C000 ;SHIFT VALUE AT $C000 LEFT
```

(16 bit divide by 2)

```
A%=A%/2

LSR $C001 ;SHIFT HIGH BYTE VALUE AT $C001
ROR $C000 ;ROTATE LOW BYTE VALUE AT $C000
```

(Blank screen to border color)

```
POKE 53265,PEEK(53265) AND 247

LDA $D011 ;LOAD ACCUMULATOR WITH VALUE
AND #$F7  ;AND WITH 247 DECIMAL
STA $D011 ;POKE VALUE BACK
```

(Position cursor to Row,Col)

```
POKE 783,PEEK(783) AND 254
POKE 781,12: POKE782,12
SYS 65520

CLC       ;CLEAR CARRY FLAG
LDX #$0C  ;LOAD X WITH 12 DECIMAL
LDY #$0C  ;LOAD Y WITH 12 DECIMAL
JSR $FFF0 ;JUMP SUBROUTINE PLOT
```

By using the PSEUDO-OP code '.EQ', you can use a label to represent any absolute memory address.

EXAMPLE:

```
10 :PLOT .EQ $FFF0
20       JSR :PLOT
```

ZERO PAGE ADDRESSING
Operand Format: $hh

All but two instructions listed under absolute addressing can use zero page addressing.

Zero page refers to memory locations $00 thru $FF (0-255 decimal). These locations are called zero page because the high byte of the address is assumed to be a zero byte.

Zero page memory can also be addressed with the absolute addressing mode. However, using zero page addressing is faster in execution time and also uses only 2 bytes instead of 3.

The two instructions used in absolute addressing that are not allowed in zero page addressing are the codes JMP and JSR.

Remember, ZERO PAGE ADDRESSING MODE operates exactly the same as absolute addressing except that ZERO PAGE ADDRESSING can only address zero page memory.

Examples:

```
LDA $FF    ;LOAD A WITH VALUE IN LOCATION
           ;$FF (ZERO PAGE)

LDA $00FF  ;LOADS THE ACCUMULATOR WITH THE
           ;VALUE IN LOCATION $FF (ABSOLUTE)
```

Zero page memory locations are used by the KERNAL OPERATING SYSTEM and the BASIC INTERPRETER. So, caution must be used when using zero page memory from BASIC.

If you have a COMMODORE 64 PROGRAMMER'S REFERENCE GUIDE, turn it to page 310. This is the memory location map. It shows you what memory locations BASIC uses for many of its operations.

As you can see, almost all of zero page memory is used up except for locations $02 $FB-FE. How then can you coexist with BASIC and use more then 5 zero page locations? Simple, just save the values from the locations someplace else and then restore them before returning to BASIC.

Remember, some locations are SPECIAL and cannot be used during BASIC execution. For instance, locations $D1 and $D2 contain the pointer to the current screen line address. If your assembly language used this memory for storing data it would change by the operating system during INTERRUPT PROCESSING if you changed the cursor position by printing something on the screen. Locations $A0-$A2 constantly change because these locations are the Real-Time Jiffy Clock. BASICs TI variable.

IMPLIED ADDRESSING

There are 25 Implied instructions in the 6502/6510 instruction set. Implied instructions need no operand and assemble to one byte codes. Here are the IMPLIED instructions:

BRK BReaK
CLD CLear Decimal flag
CLV CLear oVerflow flag
DEY DEcrement index Y
INY INcrement index Y
PHA PusH Accumulator
PLA PulL Accumulator
RTI ReTurn from Interrupt
SEC SEt Carry flag
SEI SEt Interrupt flag
TAY Transfer A to Y
TXA Transfer index X to A
TYA Transfer Y to A
CLC CLear Carry flag
CLI CLear Interrupt
DEX DEcrement index X
INX INcrement index X
NOP No OPeration code
PHP PusH Processor
PLP PulL Processor
RTS ReTurn from Sub
SED SEt Decimal flag
TAX Transfer A to X
TSX Transfer Stack to X
TXS Transfer X to Stack

INDIRECT ABSOLUTE

Only one instruction uses this addressing mode-the JuMP instruction. With this mode you can create functions similar to the ON GOTO command in BASIC.

For example, a BASIC program might have the logic:

```
10 ON A GOTO 20,30
```

Using assembly language you could write:

```
05:JMPTBL
10        .SI :FN20      STORE ADDRESS
12        .SI :FN30      STORE ADDRESS
13        ASL A          ;MULTIPLY BY 2
14        TAY            ;TRANSFER TO INDEX Y
15        LDA :JMPTBL,Y  ;LOW BYTE ADDRESS
16        STA $FB        ;STORE LOW BYTE
17        LDA :JMPTBL+$01,Y ;HIGH BYTE
18        STA $FC        ;STORE HIGH BYTE
19        JMP ($00FB)    ;GO TO ADDRESS
```

Absolute Indexed,X Addressing
Absolute Indexed,Y Addressing
Operand Format: $hhhh,X
Operand Format: :label,X

Fifteen instructions can use Absolute Indexed,X addressing. Eight of these instructions can use both Absolute,X or Absolute,Y Addressing. The other seven instructions can use Absolute,X Addressing only, and one more instruction can use Absolute,Y Addressing only. Here are the instructions:

Absolute,X and Absolute,Y	Absolute,X	Absolute,Y
ADC ADd with Carry	**ASL** Accum Shift Left	**LDX** load X
AND AND mem w/accum	**DEC** DECrement	
CMP CoMPare mem w/Acc	**INC** INCrement	
EOR Exclusive OR	**LDY** LoaD Y	
LDA LoaD Accum	**LSR** Logical Shift Right	
ORA OR mem w/Accum	**ROL** ROtate Left	
SBC SuBtract w/Carry	**ROR** Rotate Right	
STA STore Accum		

Absolute indexed instructions are used to access memory locations in a manner similar to a BASIC array. The X or Y Register can equal values from $00 to $FF (0 to 255 decimal).

Example: To print the word HELLO

```
05          ORG $C000
10 :TABLE
11          .DW "HELLO"  ;STRING
20          .DB $00      ;END BYTE
30          LDY #$00     ;INITIALIZE INDEX
40 :LOOP
50          LDA :TABLE,Y ;LOAD BYTE
60          BEQ :DONE    ;CHECK IF 0
70          JSR $FFD2    ;OUTPUT BYTE
80          INY          ;INCREMENT Y
90          BNE :LOOP    ;CONT. LOOP
95 :DONE
99          BRK          ;STOP
```

Remember, this mode is NOT the same as a label expression such as: LDA :LABEL+$03. The +$03 is called the expression. This expression cannot be changed during execution whereas Indexed Addressing uses one of the Registers of the 6510 processor that can be INCremented or DECremented. You may of course use an expression with Indexed Addressing.

LDA :LABEL+$03,Y is valid

Zero Page Indexed Addressing
Operand Format $hh,X
Operand Format $hh,Y

Sixteen instructions can use ZERO PAGE INDEXED,X Addressing. Two instructions can use ZERO PAGE INDEXED,Y. This mode of addressing works the same as Absolute Indexed Addressing, with the exception that only values $00 thru $FF can be used as the Operand. The instructions are listed below.

Zero Page Indexed,X	Zero Page Indexed,Y
ADC ADd with Carry	**LDX** LoaD X
AND AND mem with Accumulator	**STX** STore X
ASL Accum Shift Left	
CMP CoMPare mem with Accum	
DEC DECrement memory location	
EOR Exclusive OR memory with Accumulator	
INC INCrement memory	
LDA LoaD Accumulator	
LDY LoaD Y register	
LSR Local Shift Right	
ORA OR memory with Accumulator	
ROL ROtate memory Left	
ROR ROtate memory Right	
SBC SuBtract memory from Accumulator w/Carry	
STA STore Accumulator in memory	
STY STore Y register in memory	

Zero Page Indexed Addressing is faster in execution time then Absolute Indexed Addressing and only takes two bytes to specify the instruction and Operand.

Here is a routine using Zero Page Indexed Addressing to save the contents of 8 bytes of Zero page memory, so your machine language program can make use of these memory locations without affecting the BASIC storage pointers.

```
10          ORG $C000
15 :SPACE
20          .DS $0008     RESERVE MEMORY SPACE
25 :SAVE                  PROGRAM START
30          LDX #$07      INITIALIZE INDEX
35 :LOOP                  LABEL
40          LDA $2B,X     ZERO PAGE INDEX,X
45          STA :SPACE,X  ABSOLUTE INDEX,X
50          DEX           DECREMENT X
55          BPL :LOOP     BRANCH IF X>=0
60          RTS           RETURN
```

By reversing the lines 40 and 45, you can restore the values before returning to BASIC from a SYS call.

Indexed Indirect Addressing
Operand Format (\$hh,X)

Indexed indirect addressing is a combination of two addressing modes, indexed addressing and indirect addressing. Eight instructions can use this addressing mode. They are:

ADC ADd with Carry
CMP COmPare mem w/accum
LDA LoaD Accum from mem
SBC SuBtract w/Carry
AND AND mem w/accum
EOR Exclusive-OR mem w/Acc
ORA OR mem w/Accum
STA STore Accum in memory

This addressing mode is never once used in the entire BASIC INTERPRETER or the KERNAL OPERATING SYSTEM of the Commodore 64. I have never found a use for this addressing mode either. However, I will explain how it works and maybe you will find some use for it.

Example:

```
LDA ($00,X)
```

The zero page address in the instruction above ($00) is added to the value of the X register.

If the X register contains the value $2B, a byte would be loaded to the accumulator from the memory address contained in zero page memory $2B and $2C.

To access the next byte, you could increment zero page location $2B (Low byte)

By incrementing the X register twice, you could access a byte in memory from the address contained in zero page memory $2D and $2E.

Accumulator Addressing
Operand format: A

This addressing mode assembles to a one byte code just like IMPLIED addressing. The four instructions that can use this mode are:

ASL Accumulator Shift Left
LSR Local Shift Right
ROL ROtate Left
ROR ROtate Right

Example:

```
ROL A   ;ROtate Accumulator Left
```


Indirect Indexed Addressing
Operand Format (\$hh),Y

This addressing mode is, I think, the most useful. The same eight instructions described on page 23 can also be used in this addressing mode.

Example:

```
LDA ($2B),Y
```

The accumulator is loaded from the memory location contained in zero page memory $2B and $2C **plus** the value of the Y register.

If memory location $2B contains a **$01** and location $2C contains a **$08** and the Y register contains a $00, then the accumulator would be loaded with a byte from memory location $0801. By incrementing the Y index register, you can access the next 255 bytes of memory.

Example to print the word "HELLO"

```
10        .XY :WORD      pseudo-op code
20        STX $FB        store low byte
30        STY $FC        store high byte
35        LDY #$00       initialize Y
40 :LP                   label
50        LDA ($FB),Y    get ASCII character
60        BEQ :DONE      is it zero?
70        JSR $FFD2      no, output character
80        INY            increment index
90        BNE :LP        branch to :LP
92 :DONE                 all done
95        RTS            return
96 :WORD                 label
97        .DW "HELLO"    ASCII code
98        .DB $00        zero byte
```

Many times you will have tables of data you will need to access. To be able to access more then 256 bytes, you can increment the zero page memory locations high byte pointer.

Example:

```
10 :LOOP
20         LDA ($2B),Y
30         JSR $FFD2
40         INY
50         BNE :LOOP
60         INC $2C       increment high byte
70         BNE :LOOP
```

Relative Addressing
Operand Format $hhhh
Operand Format :LABEL

All relative instructions are branching instructions. They direct program flow like the BASIC command GOTO. Branching instructions differ from the absolute instruction JMP in these ways:

1. Relative instructions assemble to two bytes.

2. Relative instructions are limited to their addressing range.

3. Relative instructions test the status flags and branch only if a specified condition is met.

There are eight relative branching instructions. They are:

BCC Branch if the Carry flag is Clear
BCS Branch if the Carry flag is Set
BEQ Branch if the Zero flag is set
BMI Branch if the Negative flag is set
BNE Branch if the Zero flag is clear
BPL Branch if the Negative flag is clear
BVC Branch if the oVerflow flag is Clear
BVS Branch if the oVerflow flag is Set

Most machine language instructions, when executed, affect a bit or bits in the STATUS Register. This register is eight bits long. The Branch instructions test these flags like the IF THEN logic in BASIC. Example: To test a byte in the accumulator to find out if it is greater then zero and less then 128

```
LDA ($2B),Y
BEQ :EQUAL
BMI :NEGATIVE
fall through if A>0 and A<128
```

Pages 26 thru 29 show what flags are affected for every instruction. Try using the WALK program to see the affect of the flags while executing programs.

Branching instructions can only branch forward 127 bytes and backwards 128 bytes. If a branch is out of range, IEA will tell you during assembly.

Use a branch instruction instead of a jump instruction whenever a known condition exists such as in the program on page 24. After the INY, the BNE instruction is used because we know that the Y register will never be greater then six.

NOTATION

A	Accumulator
X,Y	Index registers
N	Negative flag
V	Overflow flag
B	BRK flag
D	Decimal mode flag
I	IRQ disable flag
Z	Zero flag
C	Carry flag
hh	8 bit hex value
hhhh	16 bit absolute address

ASSEMBLY FORM	OP CODE	N	V	B	D	I	Z	C	NO. BYTES
ADC #$hh	$69	N	V				Z	C	2
ADC $hh	$65	N	V				Z	C	2
ADC $hh,X	$75	N	V				Z	C	2
ADC $hhhh	$6D	N	V				Z	C	3
ADC $hhhh,X	$7D	N	V				Z	C	3
ADC $hhhh,Y	$79	N	V				Z	C	3
ADC ($hh,X)	$61	N	V				Z	C	2
ADC ($hh),Y	$71	N	V				Z	C	2
AND #$hh	$29	N					Z		2
AND $hh	$25	N					Z		2
AND $hh,X	$35	N					Z		2
AND $hhhh	$2D	N					Z		3
AND $hhhh,X	$3D	N					Z		3
AND $hhhh,Y	$39	N					Z		3
AND ($hh,X)	$21	N					Z		2
AND ($hh),Y	$31	N					Z		2
ASL A	$0A	N					Z	C	1
ASL $hh	$06	N					Z	C	2
ASL $hh,X	$16	N					Z	C	2
ASL $hhhh	$0E	N					Z	C	3
ASL $hhhh,X	$1E	N					Z	C	3
BCC $hhhh	$90								2
BCS $hhhh	$B0								2
BEQ $hhhh	$F0								2
BIT $hh	$24						Z		2
BIT $hhhh	$2C						Z		3
BMI $hhhh	$30								2
BNE $hhhh	$D0								2
BPL $hhhh	$10								2
BRK	$00			B		I			1
BVC $hhhh	$50								2
BVS $hhhh	$70								2
CLC	$18							C	1
CLD	$D8				D				1
CLI	$58					I			1
CLV	$B8		V						1

ASSEMBLY FORM	OP CODE	N	V	B	D	I	Z	C	NO. BYTES
CMP #$hh	$C9	N					Z	C	2
CMP $hh	$C5	N					Z	C	2
CMP $hh,X	$D5	N					Z	C	2
CMP $hhhh	$CD	N					Z	C	3
CMP $hhhh,X	$DD	N					Z	C	3
CMP $hhhh,Y	$D9	N					Z	C	3
CMP ($hh,X)	$C1	N					Z	C	2
CMP ($hh),Y	$D1	N					Z	C	2
CPX #$hh	$E0	N					Z	C	2
CPX $hh	$E4	N					Z	C	2
CPX $hhhh	$EC	N					Z	C	3
CPY #$hh	$C0	N					Z	C	2
CPY $hh	$C4	N					Z	C	2
CPY $hhhh	$CC	N					Z	C	2
DEC $hh	$C6	N					Z		2
DEC $hh,X	$D6	N					Z		2
DEC $hhhh	$CE	N					Z		3
DEC $hhhh,Y	$DE	N					Z		3
DEX	$CA	N					Z		1
DEY	$88	N					Z		1
EOR #$hh	$49	N					Z		2
EOR $hh	$45	N					Z		2
EOR $hh,X	$55	N					Z		2
EOR $hhhh	$4D	N					Z		3
EOR $hhhh,X	$5D	N					Z		3
EOR $hhhh,Y	$59	N					Z		3
EOR ($hh,X)	$41	N					Z		2
EOR ($hh),Y	$51	N					Z		2
INC $hh	$E6	N					Z		2
INC $hh,X	$F6	N					Z		2
INC $hhhh	$EE	N					Z		3
INC $hhhh,X	$FE	N					Z		3
INX	$E8	N					Z		1
INY	$C8	N					Z		1
JMP $hhhh	$4C								3
JMP ($hhhh)	$6C								3
JSR $hhhh	$20								3
LDA #$hh	$A9	N					Z		2
LDA $hh	$A5	N					Z		2
LDA $hh,X	$B5	N					Z		2
LDA $hhhh	$AD	N					Z		3
LDA $hhhh,X	$BD	N					Z		3
LDA $hhhh,Y	$B9	N					Z		3
LDA ($hh,X)	$A1	N					Z		2
LDA ($hh),Y	$B1	N					Z		2
LDX #$hh	$A2	N					Z		2
LDX $hh	$A6	N					Z		2
LDX $hh,Y	$B6	N					Z		2
LDX $hhhh	$AE	N					Z		3
LDX $hhhh,Y	$BE	N					Z		3

ASSEMBLY FORM	OP CODE	N	V	B	D	I	Z	C	NO. BYTES
LDY #$hh	$A0	N					Z		2
LDY $hh	$A4	N					Z		2
LDY $hh,X	$B4	N					Z		2
LDY $hhhh	$AC	N					Z		3
LDY $hhhh,X	$BC	N					Z		3
LSR A	$4A	0					Z	C	1
LSR $hh	$46	0					Z	C	2
LSR $hh,X	$56	0					Z	C	2
LSR $hhhh	$4E	0					Z	C	3
LSR $hhhh,X	$5E	0					Z	C	3
NOP	$EA								1
ORA #$hh	$09	N					Z		2
ORA $hh	$05	N					Z		2
ORA $hh,X	$15	N					Z		2
ORA $hhhh	$0D	N					Z		3
ORA $hhhh,X	$1D	N					Z		3
ORA $hhhh,Y	$19	N					Z		3
ORA ($hh,X)	$01	N					Z		2
ORA ($hh),Y	$11	N					Z		2
PHA	$48								1
PHP	$08								1
PLA	$68	N					Z		1
PLP	$28	N	V	B	D	I	Z	C	1
ROL A	$2A	N					Z	C	1
ROL $hh	$26	N					Z	C	2
ROL $hh,X	$36	N					Z	C	2
ROL $hhhh	$2E	N					Z	C	3
ROL $hhhh,X	$3E	N					Z	C	3
ROR A	$6A	N					Z	C	1
ROR $hh	$66	N					Z	C	2
ROR ($hh,X)	$76	N					Z	C	2
ROR $hhhh	$6E	N					Z	C	3
ROR $hhhh	$7E	N					Z	C	3
RTI	$40	N	V	B	D	I	Z	C	1
RTS	$60								1
SBC #$hh	$E9	N	V				Z	C	2
SBC $hh	$E5	N	V				Z	C	2
SBC $hh,X	$F5	N	V				Z	C	2
SBC $hhhh	$ED	N	V				Z	C	3
SBC $hhhh,X	$FD	N	V				Z	C	3
SBC $hhhh,Y	$F9	N	V				Z	C	3
SBC ($hh,X)	$E1	N	V				Z	C	2
SBC ($hh),Y	$F1	N	V				Z	C	2
SEC	$38							1	1
SED	$F8				1				1
SEI	$78					1			1

ASSEMBLY FORM	OP CODE	N	V	B	D	I	Z	C	NO. BYTES
STA $hh	$85								2
STA $hh,X	$95								2
STA $hhhh	$8D								3
STA $hhhh,X	$9D								3
STA $hhhh,Y	$99								3
STA ($hh,X)	$81								2
STA ($hh),Y	$91								2
STX $hh	$86								2
STX $hh,Y	$96								2
STX $hhhh	$8E								3
STY $hh	$84								2
STY $hh,X	$94								2
STY $hhhh	$8C								3
TAX	$AA	N					Z		1
TAY	$A8	N					Z		1
TSX	$BA	N					Z		1
TXA	$8A	N					Z		1
TXS	$9A								1
TYA	$98	N					Z		1